(4)

Technical Document 1676
October 1989

The Adaptive Kernel Neural Network

D. J. Marchette
C. E. Priebe

NAVAL OCEAN SYSTEMS CENTER

San Diego, California 92152–5000

J. D. FONTANA, CAPT, USN
Commander

R. M. HILLYER
Technical Director

ADMINISTRATIVE INFORMATION

This work was performed by the Advanced Concepts and Development Branch, Code 421, Naval Ocean Systems Center, under the Independent Research Program, OCNR-10P, Arlington, VA 22217.

Released by
M. C. Mudurian, Head
Advanced Concepts and
Development Branch

Under authority of
J. A. Salzmann, Jr., Head
Ashore Command and
Intelligence Centers
Division

JG

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1989	Final

4. TITLE AND SUBTITLE

THE ADAPTIVE KERNEL NEURAL NETWORK

5. FUNDING NUMBERS

601152N, R00N0,
ZW13
DN 309 032

6. AUTHOR(S)

D. J. Marchette and C. E. Priebe

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Ocean Systems Center
San Diego, CA 92152-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

NOSC TD 1676

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Independent Research Program, OCNR-10P
Arlington, VA 22217

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A neural network architecture for clustering and classification is described. The Adaptive Kernel Neural Network (AKNN) is a density estimation technique closely related to kernel estimation. The accompanying learning scheme adjusts the connection weights, activation functions, and the number of nodes in the network. The network, as described here, is made up of three layers of nodes: the input layer, a kernel layer and the output, or classification layer. The AKNN retains the inherent parallelism common in neural network models. Its relationship to the kernel estimator allows the network to be understood statistically, and meaningful analysis of the internal representations and the outputs is possible.

14. SUBJECT TERMS

neural networks
density estimation
kernal estimator
mixture model

15. NUMBER OF PAGES

17

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

CONTENTS

FIGURES

INTRODUCTION

The problem of pattern recognition has been studied for many years using many techniques. A typical recognition task consists of a set of exemplars from each class. The task is to produce a system which can correctly identify the members of each class. Of course, in many applications, the classes overlap, and the best that can be done is to give a measure of the probabilities for each class. One way to do this is to estimate the probability density functions of the different classes and use these as discriminators.

Recently, the technique of neural networks has enjoyed a resurgence. These networks have a number of interesting and useful attributes: they are inherently parallel, they "learn" by example, and they "generalize," in the sense that they interpolate the function sought, rather than "memorizing" the teaching set. Some networks can continue to adapt after the initial learning, which can be useful in changing environments. However, many of the currently used neural network architectures have drawbacks. For example, backpropagation can require a large number of iterations through the training set and cannot incrementally learn. The networks are often difficult to analyze, making probabilistic information about the output difficult to obtain. Finally, network size is usually determined in an ad hoc manner.

In this paper, a neural network architecture which addresses these drawbacks is described. The Adaptive Kernel Neural Network (AKNN) requires a single pass through the data. If new training sets become available, they can be incorporated in the system without requiring retraining on the original data set. Thus, it is an adaptive system in the true sense of the word. It can run in both supervised and unsupervised modes. Finally, it creates nodes as needed, and so the network grows to the size required by the problem.

The architecture is based on the kernel estimator, a nonparametric technique for estimating the probability density function of the data. This paper gives a brief description of the kernel estimator and the modifications necessary to produce an adaptive network architecture. Finally, the network is compared to the kernel estimator for some simulated data.

KERNEL ESTIMATORS

Given a set $x_1, x_2, \ldots, x_n$ of independent, identically distributed points taken from a distribution with probability density f, the task is to estimate f. If f is known to come from a particular class of densities, there are a number of parametric and nonparametric techniques to find the best match of a member of the class to the data. Without this a priori information, we would still like to get an estimate of the density. This is what the kernel estimator is designed to do.

Perhaps the most familiar technique of density estimation is the histogram. The idea is to partition the input space into rectangles and count the number of points that fall within each rectangle. Intuitively, a point near the edge of a rectangle, or bin, should effect the estimate of the neighboring bin, while a point near the center of the rectangle should have a lesser affect on adjacent bins. In the histogram estimate, all points within a bin are treated equally and there is no effect on adjacent bins.

The naive estimator is a modification of the histogram in which a rectangle is placed at each point, and the rectangles are summed to produce the estimate. More generally, any probability density function could be used in place of the rectangles. This is the idea behind the kernel estimator. The kernel estimator for f is defined as

$$\hat{f}(\mathbf{x}) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \tag{1}$$

where K is a probability density function and h is called the bandwidth or window width. For more information on the kernel estimator, see Silverman (1986).

For the remainder of this paper, the kernel K will be the multivariate normal or Gaussian distribution. The summand then becomes

$$G(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-1/2(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)} \tag{2}$$

for d-dimensional data. Here, bold-faced characters are vectors, and Σ is the covariance matrix. For practical considerations, when d is large, Σ will usually be a diagonal matrix, rather than the full covariance matrix.

This paradigm can easily be realized as a network architecture. Figure 1 shows a univariate implementation of the kernel estimator as a neural network. Each node has as its transfer function the Gaussian (equation 2), with the covariance held locally and the mean stored as the connection weights into the node. The second, or hidden, layer computes a difference between the input and its weight vector, rather than the usual dot product. The drawback of this approach is clear. The number of nodes required by the middle layer is equal to the number of points in the training set, and so the network can become extremely large. Also, the kernel estimator should be modified to allow a different variance (bandwidth) for each kernel.

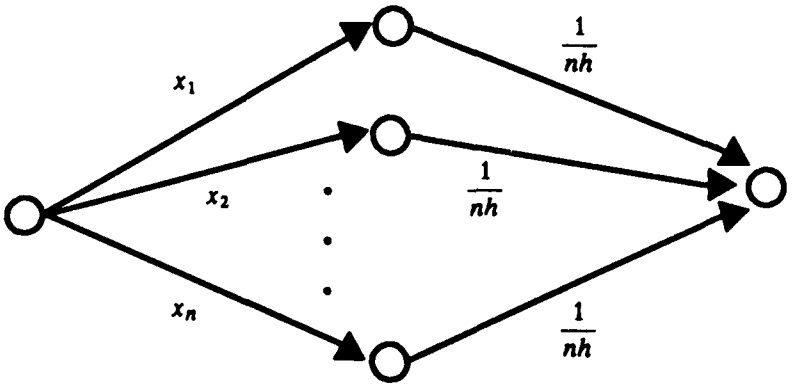


Figure 1. Network implementation of the kernel estimator.

The network implementation provides a new perspective, which allows a solution to these problems. Each node can independently determine its covariance from the data. In addition, the network can decide to add a new node based on how well the data point is covered by existing nodes. This is the basis for the AKNN.

DETERMINING NETWORK SIZE

The network size is a global property of the network and, hence, cannot be determined locally. When a data point is presented to the network, each node reports the distance from its mean to the new point. If none of the nodes are close, a new node is created with mean equal to the new point. In this manner, the network grows to a size sufficient to cover the data within a predefined distance. This is similar to the diameter-invariant cluster technique described in Sklansky and Wassel (1981). The difference is that the distance measure used is the Mahalanobis distance, which uses the covariance matrix. Since the covariance adapts to the data, as will be described below, the diameter is not really fixed in this architecture.

The algorithm for the creation of a new node is simple. A constant **c**, called the create threshold, is defined which determines the resolution of the estimate. A scaled version of the Gaussian is used as a distance measure (equation 3), which is essentially the Mahalanobis distance, exponentiated so that the values run from 0 to 1.

$$d(\mathbf{x},\mu) = e^{-1/2(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)} \quad . \tag{3}$$

If no node reports $d(\mathbf{x},\mu) > \mathbf{c}$, a new node is created. If any node reports $d(\mathbf{x},\mu) \geq \mathbf{c}$, the point is considered to be covered, and no new node is created.

A bandwidth or variance still must be chosen for the node, but since this variance will be adapted to fit the data (see the section below), this choice is not as critical as in the case of the kernel estimator. Like the create constant, the choice of initial variance controls the number of nodes created, as can be seen from the distance function d.

Although it would at first seem that **c** should be bounded between 0 and 1, this is not necessarily the case. Two interesting extremes are produced when **c** is chosen outside this range. If $\mathbf{c} > 1$, the condition $d(\mathbf{x},\mu) > \mathbf{c}$ is never satisfied, and so a node is created for each point. This is the kernel estimator described above. If $\mathbf{c} < 0$, the condition is always satisfied, after the first node is created, and so the estimate consists of a single Gaussian. In this way, the network spans the range between the parametric estimator of a single Gaussian and the nonparametric kernel estimator.

UPDATING NETWORK PARAMETERS

The network must also update the means and covariances of the nodes. It is important that this be done recursively. In many problems, it is costly or impossible to obtain large data sets for which the true classification is known. Training must be done on-line, as data are collected. The network must therefore adapt to each point, rather than allowing the luxury of repeated iterations through a fixed data set. This

allows the network to be a truly adaptive system, which continues to "learn" as each datum is presented, rather than in an off-line method requiring the storage of previous data and (often a prohibitively long) time to retrain on the data.

Two different approaches will be described. The first uses a decision similar to the create rule to decide whether a node should be updated, and then it uses recursive formulations of the sample mean formula to effect the update. In this architecture the covariance is held fixed. The second uses a Bayes-like rule to update all the nodes proportionally to the likelihood that the point came from the distribution covered by the node. The first technique is a purely local computation, but requires the user to determine another constant defining the update region. The second technique does not require this user intervention, but it is not local, requiring feedback from the output nodes.

In the first technique described, the decision to update the node is made in a manner similar to the decision to create a new node. Unlike the creation decision, this is a local decision. A constant **u**, the update threshold, is chosen, and each node for which $d(\mathbf{x},\mu) > \mathbf{u}$ updates its mean using the new point **x**. An alternative learning rule would be to update only the node with the largest value for $d(\mathbf{x},\mu)$. Note that **u** should satisfy $\mathbf{u} \leq \mathbf{c}$. For the two extremes of a single Gaussian and a kernel estimator, equality should hold.

The formula for updating the mean is an iterative version of the sample mean calculation:

$$\mu_k = \mu_{k-1} + \frac{1}{N_k}(x_k - \mu_{k-1}) \tag{4}$$

$$N_k = N_{k-1} + 1 \tag{5}$$

where subscripts indicate time.

Here, N_k is the number of points used by the node after the k^{th} input. The vector x_k is the new data point to be included in the statistics for the node. The weight on the connection from hidden node i to output (class) node j is then

$$w_{ij} = \frac{N_i}{\sum_{i->j} N_i} \quad . \tag{6}$$

The notation $i->j$ is shorthand for "node i is connected to node j." There is a recursive formulation for the computation of the weights w_{ij}, but this is unnecessary for this architecture.

It is possible to define a recursive update formula for the variance in this architecture. Unfortunately, the hard threshold of the constant **u** has the effect of ignoring points in the tail of the Gaussian, which causes a bias in the estimate of the variance. Although it might be possible to correct for this, the second learning rule, which will be described below, eliminates this problem in a more natural way. As mentioned above, this first rule, with the variance fixed, is essentially a diameter-invariant clustering technique.

It is instructive to think of the network as fitting a mixture of normals to the data. A considerable amount of work has been done in the problem of estimating the parameters of a mixture of normals, and this work can be used in the context of the network architecture. For simplicity, as above, we consider the case of estimating the probability density function of a single class. This work follows that of Titterington, Smith, and Makov (1985). First, we view the estimate as a mixture of Gaussians

$$\hat{f}(\mathbf{x}) = \sum_{j=1}^{n} \pi_j G_j(\mathbf{x}) \quad . \tag{7}$$

Here G_j is a Gaussian with mean μ_j and covariance Σ_j.

Let

$$p_k = \frac{\pi_k G_k(\mathbf{x})}{\hat{f}(\mathbf{x})} \quad . \tag{8}$$

Then

$$\mu_k = \mu_{k-1} + \frac{p_k}{N_{k-1} + p_k} (\mathbf{x}_k - \mu_{k-1}) \tag{9}$$

$$\Sigma_k = \Sigma_{k-1} + \frac{p_k}{N_{k-1} + p_k - 1} \cdot \left[\frac{p_k}{N_{k-1} + p_k} (\mathbf{x}_k - \mu_{k-1})(\mathbf{x}_k - \mu_{k-1})^T - \Sigma_{k-1} \right] \tag{10}$$

$$N_k = N_{k-1} + p_k \tag{11}$$

$$\pi_{ij} = \frac{N_i}{\sum_{i \to j} N_i} \quad . \tag{12}$$

This is a recursive technique, but is not local in the neural network sense, since it requires feedback from an output node for the computation of the p_k. This is a minor consideration, particularly if one is willing to grant neural network status to backpropagation. In this learning rule, both the mean and the covariance are updated. Points from the tails of the nodes are used in the computation of the covariance, and, in fact, if the network size is fixed, this is a recursive version of the EM algorithm used in mixture models (McLachlan and Basford 1988).

Note that once again the weights w can be determined recursively. Also, if N is a fixed constant for the mean and covariance update formulas, this has the effect of putting a window on the data, allowing the network to track slowly moving non-stationary distributions. This is a topic for future research.

EXAMPLES

Since the architecture described here models the data as a mixture of Gaussian, or normal, distributions, it is natural to consider its performance on data drawn from a mixture of Gaussians. One might hope that in this case the network would use the correct number of nodes to model the data: if the data came from a mixture of n Gaussians, the network should have n Gaussian nodes. This is too much to ask, especially without a rule for the deletion of nodes. In fact, the problem of identifying the number of components in a mixture is an unsolved problem in the theory of mixture distributions. If the number of components is known, the network size can be fixed at the appropriate amount and the parameters of the mixture will be estimated from the data.

In a typical classification task, there will be a training data set for which the correct classification is known. This allows the network to be initialized at an estimate which is consistent with the training set. This will improve the performance of the network if the training set is representative of the overall distribution. In the examples described below, however, no such training set is assumed. The network must start from scratch. This has the danger, one shared by all recursive estimators, that for small data sets the estimate is data dependent. Thus, the estimate for a data set drawn from a given distribution will be slightly different than for another data set drawn from the same distribution. The differences are noticeable on the data sets described, but not large enough to cause concern.

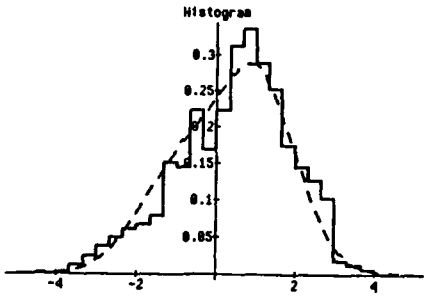
For each of the data sets described, 500 points were generated. Histograms of the data are plotted, as well as the theoretical distribution. The kernel estimator is plotted, using the optimal bandwidth described in Silverman (1986), page 40. Four data sets were used, showing both unimodal and multimodal characteristics.

A MIXTURE OF TWO GAUSSIANS

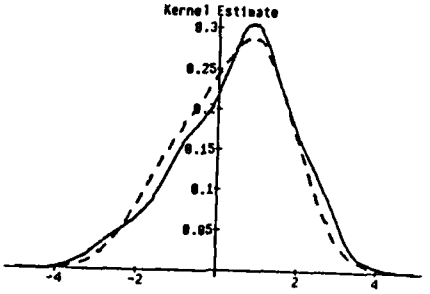
For the first data set, the data are drawn from a distribution of the form

$$x \sim \frac{1}{3}N(-1, 1) + \frac{2}{3}N(1, 1) \quad . \tag{13}$$

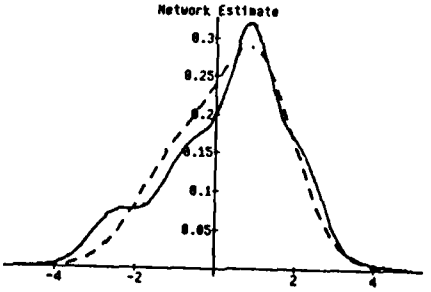
Figure 2 shows the histogram, kernel, and network estimates. As can be seen, the estimates all detect a slight bump on the left, corresponding to the smaller of the two components. The bump is smoothed in the true distribution. The network has used 9 nodes (figure 2(d)), as opposed to the kernel estimator's 500 nodes. The mode at about -2.5 in the network estimate is an artifact of the data set: the node was created near the end of the data, and not enough data have been seen since that to lower the node's weight.



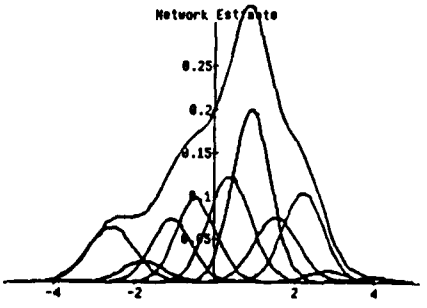
(a)



(b)



(c)



(d)

Figure 2. A unimodal two-component mixture.

The second data set is the same as the first, except the means of the components have been moved to give a bimodal distribution (figure 3).

$$x \sim \frac{1}{3}N(-2, 1) + \frac{2}{3}N(2, 1) \quad . \tag{14}$$

The network estimate has used three nodes (there are actually more, but their weights are effectively zero). Note that the network has split the smaller mode between two nodes. Given enough points, the left-most node will probably die out, leaving essentially a two-component mixture. It should be stressed that the network cannot be depended upon to find the correct mixture representation for the data, as can be seen from the first data set.

A MIXTURE OF THREE GAUSSIANS

The third data set is a combination of three components, giving a bimodal, or trimodal, distribution, depending on how one counts modes (figure 4).

$$x \sim \frac{1}{3}N(-2, .5) + \frac{1}{2}N(0, 1) + \frac{1}{6}(2, .5) \quad . \tag{15}$$

None of the estimators do a good job on this data set, though the kernel estimator is the best. Once again the network estimate has come close to estimating the correct number of components. There are five nodes, of which the smallest two will be driven down to zero with more points, though experience has shown that it can take a large number of points to "kill" these nodes.

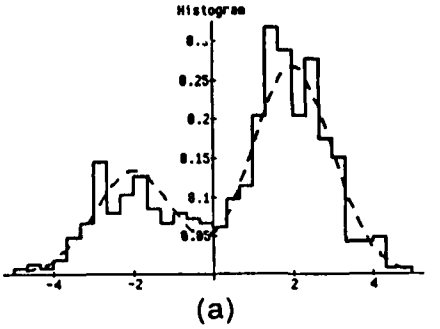
A CAUCHY DISTRIBUTION

The final example shows the effect of using the Gaussian mixture model on a distribution for which this model is incorrect. The data are drawn from a Cauchy distribution, and the network models it as four nodes (figure 5). Since the mode of the Cauchy distribution is not really a Gaussian, it is futile to try to fit it as one. To improve the fit, the network would have to create many more nodes, giving a fit similar to the kernel estimator. This can be done by changing the create threshold. Note, however, that it does make a creditable attempt to model the tails of the Cauchy distribution.

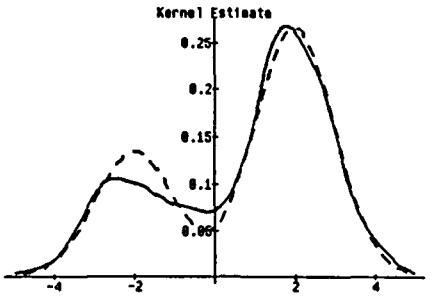
CONCLUSIONS

The AKNN is a useful tool for density estimation and its application, classification. The network can model a wide range of distributions. It is an adaptive system, and so can be used in situations where the system must continue to modify its internal representation as data are presented. It can learn the network size, given an estimate of the smoothness and composition of the density to be approximated. Unlike many other network models, training time is not an issue for the AKNN. Therefore, this network is applicable anytime the goal is classification via density estimation.

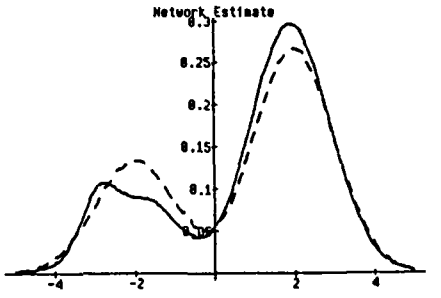
Due to its close association with statistical pattern recognition techniques (kernel estimators and mixture models) and recursive learning procedures, the AKNN is superior to conventional neural network architectures in many respects.



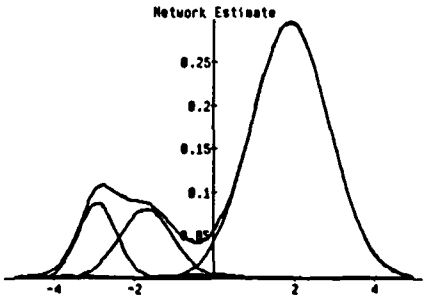
(a)



(b)

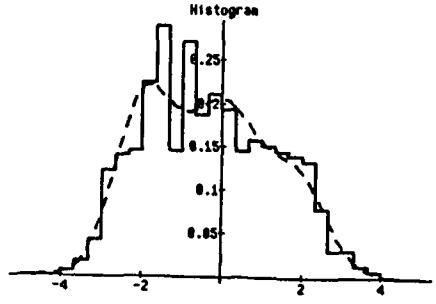


(c)

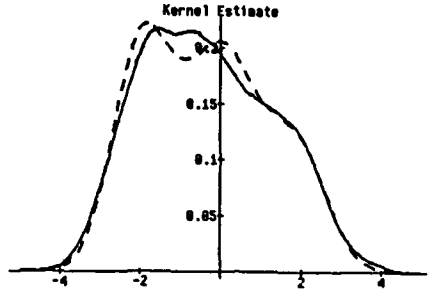


(d)

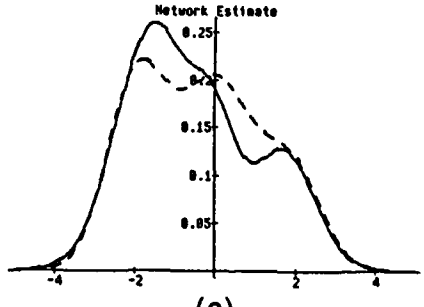
Figure 3. A bimodal two-component mixture.



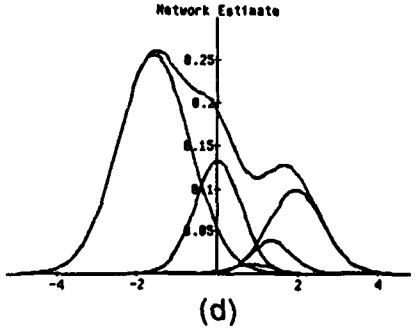
(a)



(b)

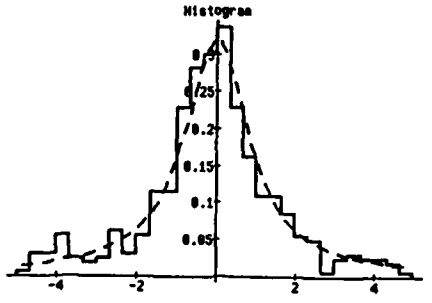


(c)

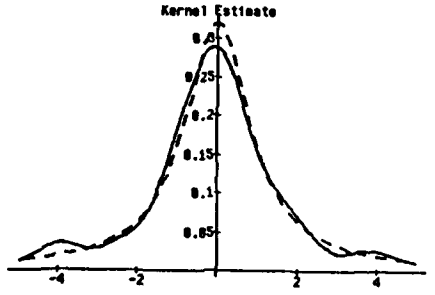


(d)

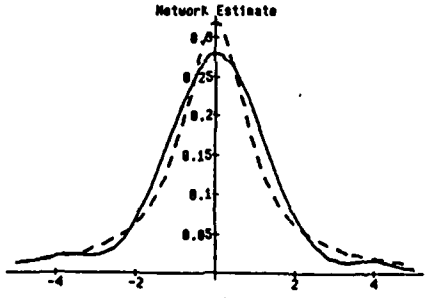
Figure 4. A trimodal two-component mixture.



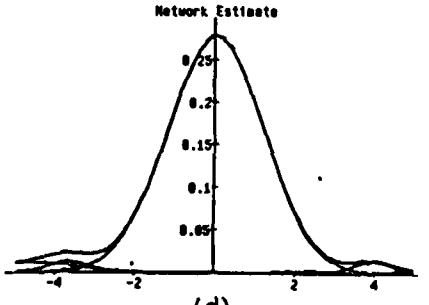
(a)



(b)



(c)



(d)

Figure 5. A Cauchy distribution.

REFERENCES

McLachlan, G. J., and K. E. Basford. 1988. *Mixture Models: Inference and Applications to Clustering*, Marcel Dekker, New York.

Silverman, B. W. 1986. *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, New York.

Sklansky, J., and G. N. Wassel. 1981. *Pattern Classifiers and Trainable Machines*, Springer-Verlag, New York.

Titterington, D. M., A. F. M. Smith and U. E. Makov. 1985. *Statistical Analysis of Finite Mixture Distributions*, John Wiley, New York.